VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

Jindřich Byrtus

Bakalářská práce

Vedoucí práce: Ing. David Seidl

Ostrava, 2021

Abstrakt

Tato práce je o mé absolvované odborné praxi ve firmě TietoEVRY. Cílem je stručně představit a přiblížit firmu jako takovou, seznámit čtenáře s projektem, na kterém jsem pracoval a rozebrat pozici, kterou jsem vykonával. Dále bych rád představil projekty a úkoly s jejich řešeními, které jsem měl na starosti a za které jsem zodpovídal. V neposlední řadě jsem shrnul uplatněné znalosti, které jsem získal v době studia, a znalosti, které jsem získal v období vykonávání bakalářské praxe.

Práce taktéž popisuje mou pracovní pozici jakožto softwarového vývojáře a testovacího inženýra, kterou jsem vykonával v týmu Network Management Automation v rámci divize Tieto Smart Utilities, a jednotlivé úkoly a problémy, se kterými jsem se musel vypořádat. Tyto úkoly a problémy zahrnovaly hledání chyb v rámci testování, zodpovědnost za kvalitu dodaného řešení při instalaci na zákaznická prostředí, odstraňování chyb, vývoj nových funkcí a optimalizace či vylepšování těch stávajících a v neposlední řadě vývoj validační utility, která slouží k odhalování chyb elektrické sítě prezentované v databázi

Klíčová slova

vývoj, testování, automatizace, optimalizace, Network Management Automation, TietoEVRY

Abstract

This thesis is about my completed professional practice done in company TietoEVRY. Target of this thesis is briefly acquaint and present closer look on the company, to acquaint the reader with the project, which I have been working on, and tell about the position I held. I would also like to present projects and tasks with their solutions for which I was responsible. Last but not least I would like to summarize the applied knowledge, which I have acquired during my studies, and knowledge, which I have gained in the period of professional practice.

Thesis also describes my job postition as a Software Developer and Test Engineer, which I held, in Network Management Automation team under Tieto Smart Utilites division, and individual tasks and issues, which I had to solve. This tasks and issues include finding bugs during testing, responsibility for the quality of the delivered software after installation on customer enviroments, repairing bugs, development of new functionalities and optimalization or improvement of the existing ones and last but not least validation utility, which servers as validation of electrical network presented in database.

Keywords

development, testing, optimalization, Network Management Automation, TietoEVRY

Poděkování

Rád bych na tomto místě poděkoval všem svým kolegům a vyučujícím, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Obsah

Seznam použitých zkratek a symbolů

NMA	–	Network Management Automation
TSU	–	Tieto Smart Utilities
NIS	–	Network Investment System
PG	–	Power Grid
WPF	–	Windows Presentation Foundation
API	–	Application Programming Interface
ORM	–	Object Relational Mapping
IIS	–	Internet Information Services

Seznam obrázků

Kapitola 1

Úvod

Ve firmě TietoEVRY vykonávám odbornou stáž od počátku třetího semestru a jednou z několika důvodu mé snahy o získání stáže byla možnost vykonat odbornou praktickou bakalářskou praxi a napsat tak bakalářskou práci z praxe.

V několika prvních částech práce popisuji společnost, jako takovou, tým, ve kterém jsem praxi vykonával, produkty, se kterými jsem se setkal a které tým vyvíjí včetně hlavní aplikace NIS, technologie, které jsem využíval a roli, jež jsem zastával.

Dále v práci líčím, jak probíhá vývoj v rámci NMA, popisuji okrajově architekturu aplikací, se kterými jsem se setkal a v neposlední řadě se snažím vysvětlit, na jakých úkolech jsem pracoval a jakým problémům jsem v rámci praxe čelil.

Práce obsahuje informace o odborných investigacích různých chyb a jejich oprav, o snaze zefektivnit práci testerské části týmu a o implementaci silného nástroje pro validaci elektrické sítě a jejich částí.

V závěru shrnu využité a chybějící znalosti ze studia, můj přínos pro tým a celkově zhodnotím mou odbornou bakalářskou praxi, kterou jsem vykonal.

Kapitola 2

O společnosti

TietoEVRY je jedním z největších gigantů mezi skandinávskými IT společnostmi, které poskytují různorodé IT služby. Působí na trhu s aplikacemi a službami v různých kategoriích jako je automotive, vzdělávání, energie, finanční služby, telekomunikace či zdravotnictví. Díky inovativnímu přístupu a technologickým vizím se TietuEVRY daří inspirovat a zapojovat své zákazníky do hledání nových způsobů zefektivnění jejich podnikatelských činností. [1, 2, 3]

Od roku 2004 má TietoEVRY své softwarové centrum v Ostravě a s více než 2 500 zaměstnanci je jedním z největších zaměstnavatelů v oblasti IT v rámci České Republiky. Od roku 2015 se Ostravská pobočka TietoEVRY umisťuje na jedněch z nejvyšších pozic v rámci studentského žebříčku TOP ZAMĚSTNAVATELÉ. [4]

2.1 NMA tým

Vzhledem k tomu, že se energetická síť stále rozrůstá, je čím dál tím složitější mít přehled nad rozložením sítě jako takové, řešit případné výpadky co nejrychleji či plánovat různá další rozšíření na základě mnoha faktorů.

Network Management Automation tým je zodpovědný za kompletní vývoj softwaru, který bude řešit veškeré výše zmíněné problémy a zefektivní tak práci energetických společností ve Finsku, Norsku či Švédsku.

2.2 Hlavní produkty NMA

2.2.1 NIS

Komplexní webový geografický informační systém , tedy webový mapový software využívající balík nástrojů a aplikací ArcGIS, který má na starost dokumentování stávající energetické sítě, plánování její renovace nebo dalšího rozšiřování a plánování údržby a oprav sítě. Tento informační systém poskytuje funkcionality nejen k dokumentaci sítě, ale také umožňuje komplexní výpočty nad elektrickou sítí, zobrazování reálného stavu sítě na základě poskytnutých dat z Data Management Systému, exporty / importy dat do jiných aplikací či vytváření hlášení pro státní organizace.

2.2.2 PGField

PGField je aplikace, která je určena k použití na přenosných zařízeních a umožňuje práci v offline režimu. Hlavním účelem je tedy detailní práce v terénu. Příkladem jsou například letové kontroly terénu, plánování výseků vegetace, přesná lokalizace elektrických prvků pomoci GPS souřadnic a mnoho dalšího. Plán nové sítě nebo údržby navržený v NISu je možné přenést do PGFieldu, zde modifikovat a poté jej opět přenést zpět do NISu.

2.2.3 Project & Budgeting

Pomocí aplikace Project & Budgeting je zákazník schopen hlídat finanční a časový rámec jednotlivých projektů a je také schopen určit cenu plánů navržených v NISu na základě komplexních konfigurovatelných pravidel. Skrze aplikaci lze na základě dat z aplikace NIS spočítat výslednou cenu materiálu, odhad ceny za práci a mnoho dalšího. Pomocí aplikace je zákazník schopen vizualizovat od kdy do kdy se výstavba či údržba bude realizovat a zkrátka kolik co bude stát.

Aplikace obsahuje kromě jiného i integraci na speciální systém pro správu úkolů, ze kterého získává data o stavu daného úkolu. Může tedy zjistit, jestli se na úkolu už pracuje, je hotový nebo je v jakémkoliv jiném stavu.

2.2.4 GPT

Tento šikovný nástroj je určen k dlouhodobému plánování investic do elektrické sítě například 10 nebo 15 let do budoucnosti. Nástroj je schopen provádět statistické výpočty pomocí kalkulačního jádra, které využívá i aplikace NIS, tvořit tematizace a vizualizace, nebo například vytvořit mapovou službu pro NIS. Díky tomuto nástroji se zákazník může rozhodnout, kde bude nejlepší investovat, aby se investice v řádech let vrátila.

2.3 Pracovní pozice

Před začátkem třetího semestru jsem se ucházel o místo na stáži ve společnosti TietoEVRY, kde jsem byl přijat na pozici manuálního testera. Během několika týdnů jsem měl za úkol obsáhnout určitý výčet funcionalit, které NIS a další aplikace obsahují. Nedílnou součástí bylo začít se orientovat ve verzované geodatabázi a rozumět tak plně procesům, které se s daty děly.

Zhruba po dvou měsících jsem byl schopen zastupovat testery v průběhu regresního testování či testování na zákaznických prostředích po doručení nové verze. Byl jsem zodpovědný za kvalitu určité části dodaného řešení.

Později jsem dostal na starost testování oprav chyb, investigaci chyb v rámci zákaznické podpory a následně jejich opravy, pokud se jednalo o chybu v datech či v konfiguraci služeb a nástrojů, které k aplikaci patří, testování nových funkcionalit a na základě získaných informací vytvářet balíčky testů pro ostatní testery.

Po půl roce jsem dostal příležitost podílet se na vývoji aplikace a začal jsem opravovat jednoduché chyby na frontendu či backendu. Jednou z mých prvních aplikací byl nástroj pro testery, který jsem vytvořil za účelem zefektivnění práce při opravách XML souborů určených pro transfer dat z plánů. Postupně jsem začal dostávat příležitosti investigovat a opravovat komplexní chyby v kódu a vyvíjet nové funkce nejen v NISu, ale i v nástrojích, které úzce souvisely například s údržbou aplikace.

Mým největším projektem, který jsem měl na starosti, je bezpochyby validační nástroj, jež má za úkol hledat chyby v databázi a co nejefektivněji je zaznamenávat tak, aby bylo možné navrhovat automatické opravy na základě těchto záznamů či poskytovat zákaznickým specialistům hlášení, kde se chyby nachází, a zajistit tak integritu dat. V tomto případě jsem měl možnost úzce spolupracovat s největšími specialisty našeho týmu a vyvinout tak velice důležitý a silný nástroj.

2.4 Testování v rámci NMA

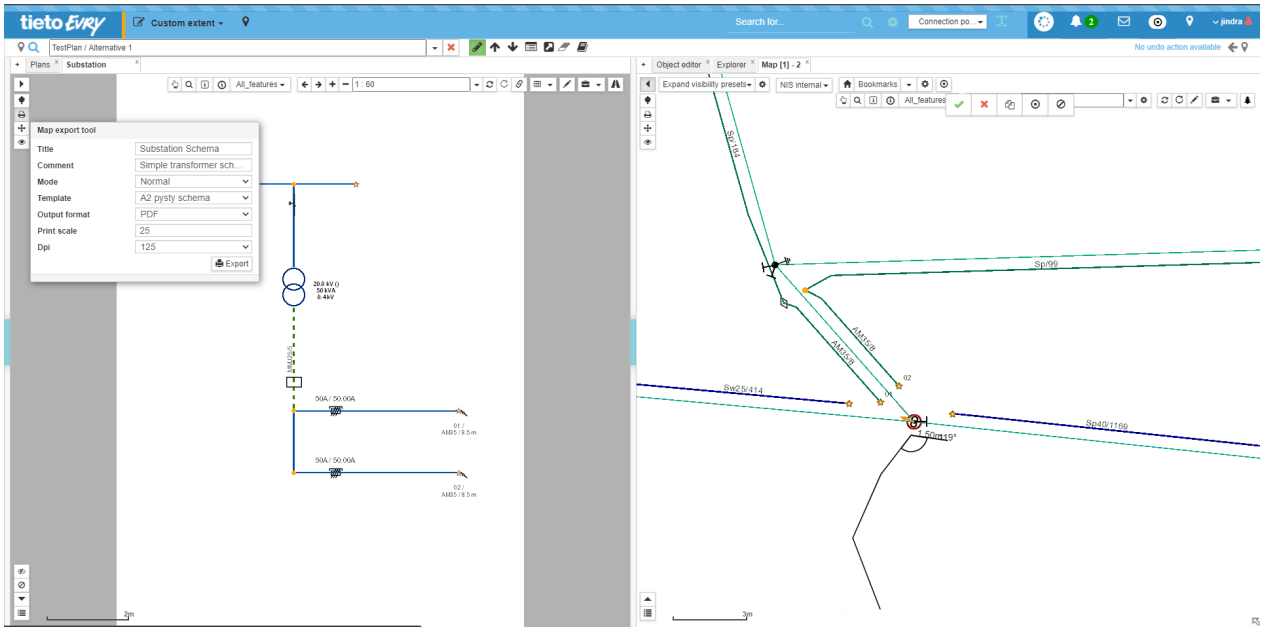
V rámci NMA je testování aplikací a nástrojů rozděleno na manuální testování a testování automatizované. Manuální testeři mají za úkol testovat záležitosti, které nelze či by bylo obtížné automatizovat. Automatické testy jako takové se dělí na testy služeb a GUI testy, které jsou exekuovány jak nad hlavní aplikací NIS tak nad aplikací Project and Budgeting.

2.5 Vývoj v rámci NMA

Veškerý vývoj produktů je koncipován do modelu SCRUM, což je jedna z agilních metodik vývoje softwaru. V rámci krátkých intervalů zvaných sprint, které trvají několik týdnů, se naplánuje, co se bude vyvíjet a jaká problematika se v nadcházejícím období bude řešit. V rámci plánování se rozdělí zdroje týmu a rozdělí se prioritizované úkoly mezi jednotlivé členy či skupiny. Kromě plánování jednotlivých iterací vývoje každý den probíhá ranní meeting, kde každý z týmu má příležitost sdělit své poznatky, požádat o pomoc či asistenci a popřípadě prioritizovat určitý problém před ostatními.

2.6 O aplikaci NIS

NIS je balíčkem nástrojů a služeb, které jsou vytvořeny pro účely zefektivnění plánování elektrické sítě jako takové nebo dále kromě jiného pro jednotlivé komplexní výpočty celé sítě, na základě kterých se zákazník může rozhodnout, jestli je třeba danou síť vylepšit o určitý elektrický prvek či určitou část sítě renovovat. Jak bylo zmíněno, aplikace je postavena na technologii ASP.NET



Obrázek 2.1: Grafické rozhraní aplikace NIS

MVC. Z pohledu architektury se jedná o spíše monolitickou aplikaci, jež je rozdělena na dvě hlavní části. První představuje single-page webovou frontend aplikaci, která komunikuje s druhou částí, jež poskytuje REST API rozhraní a implementuje back-end. Tato back-endová aplikace zaštiťuje jak přímou implementaci bussiness logiky tak komunikaci s mnohými dalšími webovými službami. Valná většina backendu je napsána v jazyce C# a využívá technologií .NET. Komunikace mezi klientem

a službami je zajištěna pomocí REST API. V rámci vývoje a práce s jednotlivými službami jsem se setkal i s C/C++ výpočetním jádrem, které se stará o výpočet elektrické sítě.

Na frontendu běží Javascript s JQuery a s knihovnami ESRI JS API 3.x , které využívají Dojo Toolkit kvůli modulárního systému AMD. Grafické rozhraní je vytvořeno s využitím technologií Razor, Bootstrap a CSS, díky kterým je možné efektivně dynamicky implementovat i složité grafické prvky.

Celá aplikace je koncipována jako jednostránková mapová aplikace, která zpracovává geografická data a využívá technologií společnosti ESRI a to sice ArcGIS Server a jeho balíček nástrojů včetně generování mapových služeb a ESRI verzování geodatabáze.

Plánování v rámci elektrické sítě je realizováno pomocí verzování. Zákazník je schopen vytvořit nový plán elektrické sítě a kreslit novou část sítě či jinak upravovat síť stávající. Tyto verze je možné exportovat a importovat do jiných plánů nebo například používat v aplikaci PGField a plánovat tak přímo v terénu. Aplikace implementuje složitou funkcionalitu nahrávání změn z/do těchto verzí, jejich každodenní aktualizaci a řešení možných konfliktů mezi verzovanými daty.

Aplikace se stále vyvíjí a je rozšiřována o nové nástroje využívající data třetích stran jak z veřejného tak soukromého sektoru. Například katastrální data, informace o podloží, městské územní plány, data z aplikací pro řízení elektrické sítě v reálném čase a mnoho dalšího.

Kapitola 3

Technologie

3.1 MSSQL

Microsoft SQL Server je robustní a kvalitní relační databázový systém, který umožňuje přehlednou a rychlou práci s daty. [5]

3.2 T-SQL

Transact-SQL je balík rozšíření pro programování v SQL. Díky němu můžeme využívat například komplexních funkcí a procedur přímo na databázovém serveru, provádět práci s daty v rámci transakcí, automatizovat určitě procesy pomoci triggerů a mnoho dalšího. [6]

3.3 ArcGIS

ArcGIS je geografický informační systém pro práci s mapami a geografickými informacemi. Informační systém vyvinula společnost ESRI a je využíván například k vytváření, úpravě a udržování různých map, k analýze mapových informací, ke kompilování geografických dat a tak dále.[7]

3.4 WiX Toolset

Balíček nástrojů pro vytváření instalátorů pro Windows instalační jádro. WiX je kromě jiného schopen instalovat SQL databáze, IIS webové aplikace, modifikovat firewall, instalovat různé balíčky jako je například .NET Core a tak dále. Nabízí tedy relativně slušnou škálu možností automatizace procesů v rámci instalace aplikace na různá prostředí. [8]

3.5 .NET Framework

Jedná se framework od společnosti Microsoft, který obsahuje velkou knihovnu tříd a funcionalit. Software naprogramován v rámci .NET Framework je spuštěn v prostředí Common Language Runtime, tedy v určitém virtuálním stroji, který se stará kromě jiného o správu paměti, bezpečnost a o zpracovávání výjimek. [9]

3.6 .NET Core

.NET Core je multiplatformním nástupcem .NET Framework a přináší mnoho výhod. Díky podpoře jiných operačních systému je práce s ním efektivnější. V listopadu 2020 byla vydána verze .NET 5.0, která plně nahradila .NET Framework a z názvu bylo odstraněno slovíčko "Core". Tato verze včetně ostatních vylepšení nabízí například podporu Arm64, tudíž programy nemusí běžet jako emulované verze x86. [9]

3.7 Entity Framework

Je ORM framework od společnosti Microsoft, který umožňuje vyšší úroveň abstrakce. Tohoto faktu dosahuje díky možnosti pro vývojáře se nezaobírat databázovými tabulkami jako takovými, ale naopak umožňuje práci se specifickými objekty a jejich vlastnostmi. Zápis kódu je díky tomu mnohem čitelnější a jednodušší. [10]

3.8 Windows Server

Windows Server je serverový operační systém, který nabízí velké množství funkcí sloužících k různým záležitostem. Z jeho funkcí můžu zmínit například IIS, tedy softwarový webový server, který slouží například k hostování webových aplikací, nebo SQL Server, který je relačním databázovým systémem. [11]

Kapitola 4

Datová struktura elektrické sítě a práce s verzemi

Abych mohl čtenáři sdělit, co jsem v průběhu své odborné praxe vykonával, musím aspoň nastínit datovou strukturu elektrické sítě, se kterou aplikace pracují, a vysvětlit nejzákladnější operace s verzemi, které využívá uživatel, aby mohl se sítí pracovat

4.1 Datová struktura elektrické sítě a relace mezi jejími prvky

Elektrickou síť v tomto případě můžeme rozdělit na prvky elektrické a neelektrické, tedy prvky, kterými neprochází elektřina a slouží například jako uchycení kabeláže či jako objekt, ve kterém se elektrické prvky nachází. Všechny prvky elektrické sítě mají svou vlastní geometrii a můžeme je nazvat prvky geometrickými

Neelektrickými prvky jsou například sloupy a jejich podpěry, cesty či výkopy, kterými kabeláž vede, stanice a rozvodny, které mají svá elektrická schémata, a další podpůrné objekty.

Mezi elektrické prvky můžeme zcela jistě zařadit rozpojovače a spínače, jističe, kabely a měděné lyžiny, elektrické výstupy, generátory, uzemnění a například odběrná místa či přípojky, které jsou objektem, kde je zákazník schopen vytvořit zátěž na síti. Elektrické prvky dělíme na linkové a bodové objekty, které mají relaci na svou topologii.

Topologii elektrické sítě tvoří všechny elektrické prvky. Bodové objekty jsou jednotlivými vrcholy a linkové objekty, jako jsou kabely, tvoří hrany topologického spojitého grafu. Každý elektrický prvek má tedy relaci na své topologické protějšky, které mají svou databázovou tabulku. Jak již výše zmíněné informace napovídají, dělíme je na uzly a linky. Na základě různých algoritmů průchodů grafem, je poté výpočetní jádro schopno vytvářet komplexní výpočty nad topologií a počítat tak různé elektrické hodnoty, jakými jsou napětí, proud, zkratový proud, počítat zátěže v závislosti na čase, vizualizovat směr napětí, zobrazovat živé a mrtvé části sítě a mnoho dalšího.

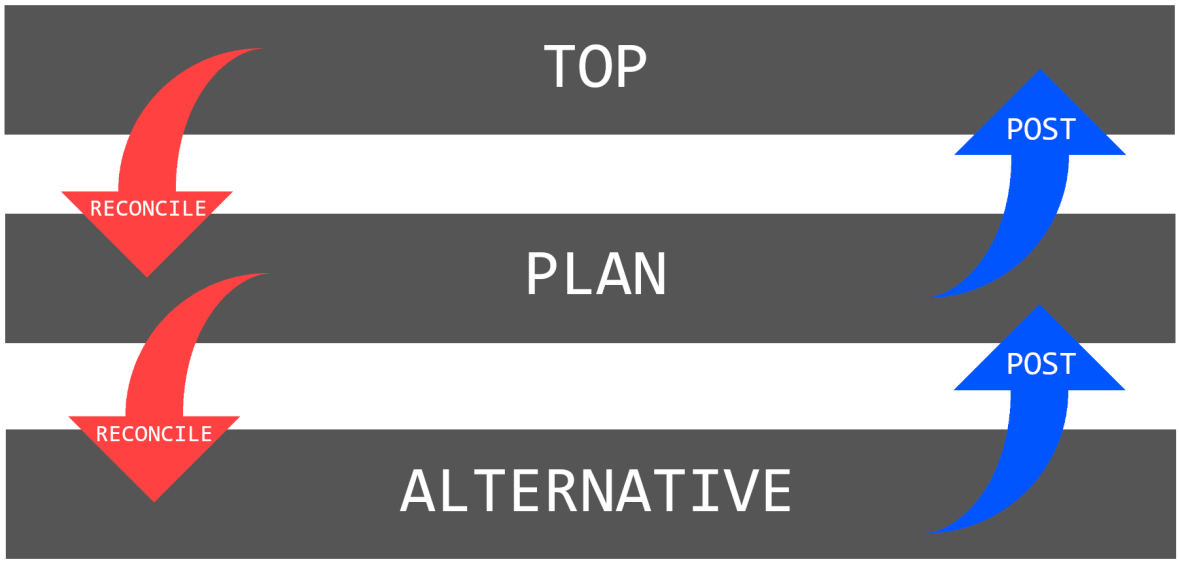
Mezi modelovanými prvky existují další relace nejen již zmíněné relace elektrických prvků na svou topologii. Mohu zmínit například odkaz měděných lyžin, spínačů a dalších zařízení na majitele elektrického schématu, tedy rozvodnu nebo elektrickou stanici. Dalšími relacemi mohou být třeba odkazy mezi kabeláží a jejich cestami, kterými kabely vedou.

Celá síť je tedy různě provázána skrz databázi pomocí různých relací mezi jednotlivými prvky.

4.2 Práce s verzemi

Aplikace implementuje několik možností, jak s jednotlivými daty pracovat a plánovat tak různé operace nad elektrickou sítí.

Jednotlivé verze lze rozdělit na tři úrovně. Nejvyšší úrovní je hlavní verze, která reprezentuje reálný stav elektrické sítě. Tuto verzi nazýváme verzí topovou, zkráceně TOP. Další úrovní jsou plánové verze, tedy jednotlivé plány, které zákazník může tvořit. Aby ale mohl zákazník plány tvořit, musí vytvořit ještě další, třetí, úroveň verze, kterou nazýváme alternativou. V rámci této alternativy, tedy alternativy plánu, je zákazník schopen s elektrickou sítí pracovat. Může v rámci ní navrhovat úpravy nebo kreslit nové části elektrické sítě. V ideálním případě by měla alternativa existovat jeden den, kdy uživatel svou celodenní práci zanese pomoci operace POST do nadřazené plánové verze a alternativa tak může zaniknout.



Obrázek 4.1: Znázornění verzí a operací nad nimi

Nad těmito verzemi aplikace implementuje složité operace, kterými jsou kromě POST a RECONCILE i další komplexní operace s daty. Pomocí operace POST je uživatel schopen zanést data z POSTované verze do verze nadřazené. Naopak pomoci operace RECONCILE je schopen verzi, pro kterou operaci provádí, synchronizovat s verzí nadřazenou. [12]

Kapitola 5

Svěřené úkoly

5.1 Validace a oprava migračních souborů formátu XML

Jedním z prvních úkolů, který mi byl svěřen, bylo validovat a opravovat soubory formátu XML, které slouží k přenosu dat mezi aplikacemi.

Díky potřebě zachovat integritu dat, nevalidní soubory nemohou být importovány do aplikace. Právě proto bylo potřeba odborného zásahu a na základě získaných informací v průběhu komplexní investigace soubory opravovat.

Časová náročnost: 2 měsíce

5.2 Analýza a manuální opravy dat elektrické sítě a jednotlivých verzí

Dalším z mnoha úkolů bylo analyzovat chyby ve verzovaných datech elektrické sítě a na základě například závislosti mezi nadřazenou verzí a podřazenou navrhnout opravu.

Nedílnou součásti této analýzy byla i investigace příčiny, která chybu mohla způsobit. Ať už se jednalo o data, která se do databáze dostala migrací dat a nebo příčinou byla chyba v kódu.

Časová náročnost: dlouhodobá práce

5.3 Implementace validační utility elektrické sítě

Na základě investigovaných chyb z historie jsem dostal za úkol implementovat utilitu, která bude mít za úkol najít a zaznamenat veškeré chyby v rámci dat elektrické sítě.

Požadavky na nástroj zahrnovaly efektivnost vykonávaných testů, protože byť sebelepší test, který běží celou noc, je k ničemu, atomičnost jednotlivých testů a srozumitelnost, tedy jednoduchost, aby aplikaci bylo relativně jednoduché kdykoliv upravit či rozšířit o další testy tak, aby jiný programátor nemusel znát dopodrobna architekturu utility.

Časová náročnost: 3 měsíce

Kapitola 6

Validace a oprava plánových souborů k exportu a importu

Aplikace NIS je nástupcem starší desktopové aplikace Power Grid, kterou již dříve vyvíjeli někteří členové našeho týmu. V průběhu vývoje vznikla potřeba přenášet data mezi různými aplikacemi. K tomuto přenosu dat bylo využito souborů .zip, které obsahují kromě jiného například fotografie a dokumenty k různým prvkům elektrické sítě a samotný XML soubor, ve kterém se nacházejí všechna data z plánů. Tyto soubory mohou zákazníci přenášet mezi aplikacemi jako jsou PGField, NIS a tak dále. Mohou je tedy exportovat či vytvořit v určité aplikaci a dále je importovat do jiné.

Pomocí různých operací je zákazník schopen způsobit různé chyby v elektrické síti. Mnohé z těchto chyb jsou zanášeny při importu XML souboru s daty plánu. Tento soubor může existovat řadu let a na jeho modifikaci spolupracovalo několik aplikací, které mohly mít podkladová data z jiných dob. Díky tomu může po importu do NISu dojít k vytvoření nekonzistence v datech a je na uživateli NISu, aby v rámci finalizace plánu tyto chyby odstranil. K tomu NIS poskytuje řadu nástrojů. Bohužel některé starší plány nebylo bez asistence podpory možné importovat.

Chyby se objevovaly třeba v atributech jednotlivých elektrických prvků, vznikaly různé duplikáty prvků nebo se objevil nevalidní unikátní identifikátor prvků, který vznikl v průběhu plánování v PGFieldu. Takovýto poškozený XML plán nemohl být importován do NISu a potažmo do databáze kvůli riziku ztráty integrity dat.

V rámci zákaznické podpory jsem se musel potýkat s investigací poškozených XML plánů a s následnou opravou i importem do aplikace.

6.1 Investigace XML plánů

Na základě zákaznického hlášení jsem musel investigovat, kde v XML plánu je chyba, opravit ji a následně importovat do aplikace, pokud si to zákazník přál.

Jedním z prvních kroků bylo získání xml plánu, který se buď nacházel jako příloha u založeného problému nebo díky své velikosti na zákaznickém prostředí. Po úspěšném získání XML plánu jsem byl schopen plán zobrazit a mohl jsem se pokusit nalézt nejzjevnější chyby pomocí vyhledávání různých klíčových slov.

Nicméně pokud byl plán obsáhlejší, tedy řádově ve statisících řádků XML souboru, nalézt takovou chybu bylo mnohdy obtížné obzvláště pokud jsem na takovou ještě nenarazil.

Dalším krokem, pokud jsem nebyl schopen chybu najít ručně, byl import do aplikace. Díky robustní kontrole importovaných dat, jsem mohl téměř s klidným svědomím plán importovat a na základě zachycených výjimek zaznamenaných v aplikačním logu jednotlivé chyby nacházet a opravovat.

6.2 Opravy XML plánů

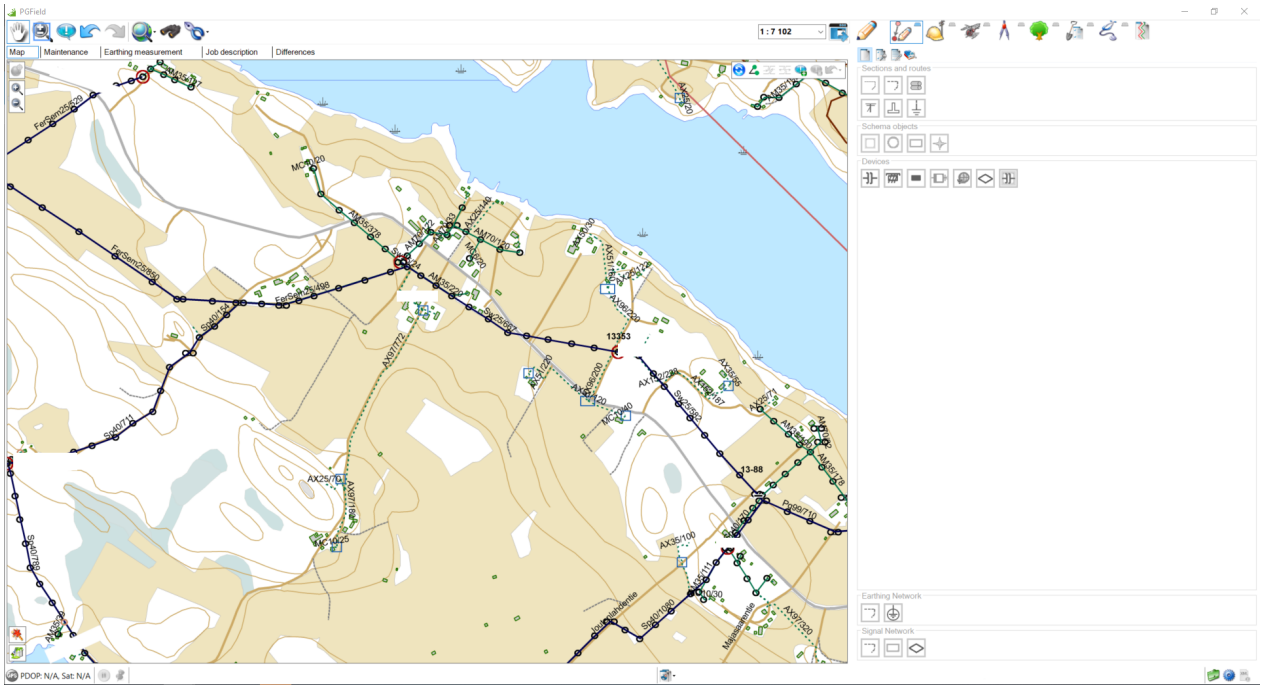
Chyby v plánech byly různorodé. Mnohdy se jednalo o nevalidní atributy či chybějící záznamy o prvcích, na které objekty měly mít relaci. Některé chyby jsem mohl odhalit obyčejným vyhledáváním klíčových slov.

Chyby tohoto typu, tedy prázdné atributy, jsem musel opravit buď na základě informací z databáze, pokud se jednalo o prvek, který tam již byl zaznamenán, a nebo prvek smazat a popřípadě sdělit zákazníkovi o který prvek se jednalo, aby jej byl schopen znova nakreslit.

Další častou chybou byly duplikáty ať už v topologických prvcích nebo geometrických prvcích jako takových. Mnohdy se jednalo o stejné prvky, které díky tomu měly stejné ID a nebyly vpuštěny kontrolou do aplikace. Po ujištění, že se opravdu jedná o naprosto přesnou kopii prvku, mohl jsem jeden z duplikátů odstranit.

Někdy se však jednalo i o mírné nuance mezi prvky například v délce nataženého kabelu a bylo potřeba investigovat na základě topologických dat či geometrických, který z duplikátů je ten správný a druhý opět smazat.

Jindy se často objevovaly chyby s nevalidní geometrií, kdy například linkový objekt, jako je měděná lyžina nebo elektrický kabel, měly nulovou délku, tedy bodovou geometrii. V naprosté většině takových problémů jsem musel daný plán načíst do aplikace PG Field a investigovat, jak daný elektrický prvek má vypadat, opravit jej a zpět exportovat.

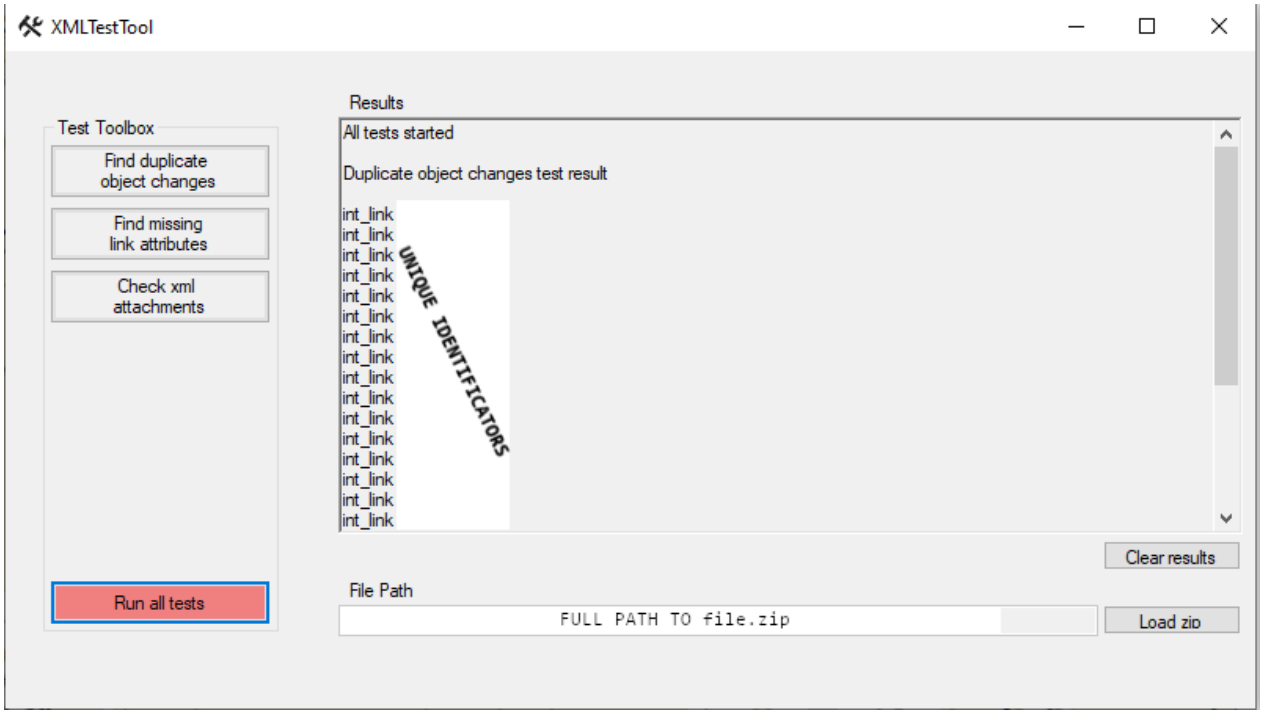


Obrázek 6.1: Grafické rozhraní aplikace PGField

Po kompletní opravě chybného XML plánu bylo konečně možné úspěšně importovat plán, pro který se vytvořila alternativa, tedy verze podřízená plánové verzi, a spustit kontrolu změn, která je implementována v aplikaci. Když jsem si byl jistý dobře odvedenou prací, mohl jsem dát vědět zákazníkovi, že s plánem může dále pracovat a že XML plán je opraven a naimportován.

6.3 Zefektivnění práce v rámci investigace chybných XML plánů

Vzhledem k tomu, že ve většině případů, stejně jako ostatní testeři, kteří této problematice rozuměli, jsem byl odkázán na import do aplikace a dlouhé čekání na výjimku v logu aplikace, která se týkala jedné chyby, rozhodl jsem se implementovat nástroj, který byl schopen většinu známých problémů odhalit a navézt tak uživatele na přesná místa v XML souboru, která byla třeba opravit.



Obrázek 6.2: Grafické rozhraní aplikace NIS

Aplikaci jsem postavil na .NET Frameworku a zvolil jsem si platformu WPF, aby obsluha aplikace byla pro uživatele přívětivější. Pomocí několika tříd z knihovny Linq, tedy zejména tříd XDocument, XElement a tak dále, jsem byl schopen XML přečíst a data, která jsem chtěl, načíst do různých kolekcí.

Nad kolekcemi, tedy jednotlivými prvky, jsem byl schopen provádět operace jako vyhledávání duplicit.

Nebo například vyhledávání nulové geometrie.

Na základě nalezených chyb jsem mohl sdělit uživateli unikátní hodnoty a informaci o dané chybě zaznamenat do textového boxu, který pro to bylo určený.

V těch horších případech, kdy XML plány byly velké řádově několik desítek megabytů a import, než byla odhalena jedna chyba, trval i půl hodiny, jsem pomocí mého nástroje dokázal odhalit všechny chyby během několika málo sekund pomocí jednoho tlačítka. Následná oprava na základě těchto informací byla velice rychlá.

Je tedy nutné zmínit, že efektivita práce se ve většině případů výše zmíněné práce markantně zlepšila. Odhalování takových chyb trvalo třeba i několik hodin. Kdežto s mým nástrojem byly testeři schopni celou záležitost včetně importu vyřešit do několika málo desítek minut.

Kapitola 7

Analýza a manuální opravy dat elektrické sítě a jednotlivých verzí

Díky nejen výše již zmíněným faktům, bylo možné do databáze zanést chyby, které bylo třeba investigovat a manuálně opravit. Někdy se jednalo o chyby vzniklé ve staré aplikaci třetí strany a zanesené datovým přenosem topové, tedy hlavní, reálné verze elektrické sítě, mezi databázemi.

Takovýto přenos probíhal v rámci noční údržby ještě v době, kdy zákazník nepoužíval NIS jako master, tedy hlavní produkční systém. V databázi jsem narážel na prvky s chybějící či nevalidní topologií, prvky s chybnou geometrií, nebo třeba topologii s relací na neexistující prvky.

Dále jsem nacházel prvky, které měly například nastavené špatně atributy a nebylo nad nimi možné provádět výpočty nebo výpočet negativně ovlivňovaly.

Mnohdy jsem musel investigovat verze, které nebylo možné aktualizovat, protože obsahovaly konflikty v datech či jiné chyby, které aktualizaci zabraňovaly.

7.1 Investigace chyb elektrické sítě

Prvním krokem bylo správně pochopit nahlášenou chybu na základě poskytnutých informací zákazníkem v založeném problému. V nejlepším případě mi zákazník poskytl název plánové verze, ve které se chyba nacházela, a například unikátní název či ID daného chybného prvku.

Následně jsem se snažil identifikovat problém v aplikaci pomocí několika připravených nástrojů, které kolegové vývojáři přesně pro takovou investigaci vyvinuli.

V případě, že nebylo zcela jasné, kde je chyba, bylo třeba pracovat přímo s databázovými daty na základě SQL dotazů, které jsem musel pro tyto účely vytvářet.

Pomocí komplexnějších SQL dotazů jsem byl chyby podobného charakteru schopen nacházet a najít tak například určitou relaci mezi nimi nebo odhalit tak chyby, které bylo třeba opravit taktéž.

```
SELECT l.[NETWORK_TYPE], n.[NETWORK_TYPE], l.[ID] as LINK_ID, l.[DEVICE_TYPE] as
    LINK_DEVICE_TYPE, l.[DEVICE_ID] as LINK_DEVICE_ID, l.[SCHEMA_OWNER_TYPE] as
    LINK_SCHEMA_OWNER_TYPE, l.[SCHEMA_OWNER_ID] as LINK_SCHEMA_OWNER_ID, n.[ID] as
     NODE_ID, n.[DEVICE_TYPE] as NODE_DEVICE_TYPE, n.[DEVICE_ID] as NODE_DEVICE_ID
    , n.[SCHEMA_OWNER_TYPE] as NODE_SCHEMA_OWNER_TYPE, n.[SCHEMA_OWNER_ID] as
    NODE_SCHEMA_OWNER_ID FROM [LINKS] l
   JOIN [NODES] n ON l.[FIRST_NODE_ID] = n.[ID]

   WHERE (l.[DEVICE_TYPE] <> <transformer_type> AND n.[DEVICE_TYPE]<> <
      transformer_type>) AND l.[NETWORK_TYPE] <> n.[NETWORK_TYPE]

UNION

SELECT l.[NETWORK_TYPE], n.[NETWORK_TYPE], l.[ID], l.[DEVICE_TYPE], l.[DEVICE_ID],
     l.[SCHEMA_OWNER_TYPE], l.[SCHEMA_OWNER_ID], n.[ID], n.[DEVICE_TYPE], n.[
    DEVICE_ID], n.[SCHEMA_OWNER_TYPE], n.[SCHEMA_OWNER_ID] FROM [LINKS] l
   JOIN [NODES] n ON l.[LAST_NODE_ID] = n.[ID]

   WHERE (l.[DEVICE_TYPE] <> <transformer_type> AND n.[DEVICE_TYPE]<> <
      transformer_type>) AND l.[NETWORK_TYPE] <> n.[NETWORK_TYPE]
```

Výpis kódu 7.1: Příklad anonymizovaného SQL dotazu pro investigaci chyb 1

Důležitým krokem, na který jsem vždy kladl důraz, byla identifikace příčiny těchto chyb a následná komplexní investigace, aby se něco podobného v nejlepším případě už nemohlo opakovat, a následná konzultace se seniorními developery v našem týmu. Pokud se jednalo o chybu v kódu, snažil jsem se problém replikovat, abych poskytl co nejpřesnější informace vývojářům a šetřil tak jejich čas.

7.2 Oprava chyb elektrické sítě

Na základě důkladné investigace, kterou jsem provedl předtím, jsem mohl danou chybu, pokud to bylo v mé kompetenci, opravit. Pokud se jednalo o chybu, která šla odstranit skrze funkcionality aplikace, použil jsem je, abych si byl jistý, že dělám správné kroky.

Rád bych uvedl jeden příklad a to sice nesoulad mezi topologií prvku a samotným prvkem. V tomto případě se jednalo o elektrický odpojovač, který se uživateli v aplikaci jevil jako zavřený, tedy průchozí, ale algoritmus průchodu grafem a následný výpočet různých elektrických veličin jej nespočítal nebo neprošel skrze něj.

Vzhledem k tomu, že vím, že výpočetní jádro pracuje na základě topologie elektrické sítě, jsem se snažil hledat chybu v topologii. Po správné identifikaci atributu, který udává stav topologické průchodnosti, jsem zjistil, že je podle topologického bodu prvek otevřený, tedy neprůchozí. Následně, na základě předchozího zjištění, jsem přišel na to, že prvek, jako takový, by měl být však zavřený. Pomocí nástrojů NISu jsem byl schopen prvek otevřít a znovu zavřít, což mělo za následek po otevření nastavení jednotného stavu jak prvku topologického tak geometrického a po zavření uvedení prvku do pro zákazníka výchozího stavu.

Následným spuštěním výpočtu sítě a zobrazením směru napětí na mapě, jsem se ujistil, že je chyba opravena a prvek včetně sítě, která na něj byla napojena, je opět průchozí.

Pomocí SQL dotazu jsem se pokusil najít další podobné prvky, které mají tento atribut rozhozený mezi geometrickým prvkem a jeho topologií a na základě zjištění dále navrhnout opravu buď pomocí hromadného SQL příkazu, kde jsem mohl prvky jednoduše přenastavit, a nebo pomocí funkcionality aplikace stejnou sérií úkonů, jež jsem provedl u prvního prvku. Efektivnějším řešením byl však správný SQL příkaz, který jsem po ujištění se, že se jedná o správně zvolené prvky na základě WHERE klauzule, mohl spustit.

```
SELECT * FROM [DEVICE] AS d
   JOIN [NODE] AS n ON n.[DEVICE_ID] = d.[ID] AND
      n.[DEVICE_TYPE] = (
         SELECT [DEVICE_TYPE] FROM [DEVICE_TYPES]
            WHERE [DEVICE_TYPE_NAME] LIKE '<device_name>'
         )
   WHERE n.[STATE] != d.[STATE];
```

Výpis kódu 7.2: Příklad anonymizovaného SQL dotazu pro investigaci chyb 2

Mnohdy však nebylo možné chyby opravit skrze aplikaci kvůli určitým pravidlům práce s elektrickou sítí, takže jsem musel na základě získaných informací navrhnout řešení a problém manuálně opravit přímo v databázi. Častým problémem, jak jsem již zmiňoval, byla třeba špatná relace mezi topologickými a geometrickými prvky. V průběhu investigací a následných oprav sítě, jsem narážel na prvky, které v mapě vypadaly naprosto správně, ale jejich výpočty, nebo například vykreslo-

vání směru napájení, správné nebylo. Na základě této skutečnosti jsem předpokládal, že se jedná o topologickou chybu. Po prozkoumání sítě jak v aplikační mapě, tak v databázi, jsem zjistil, že topologie prvků je odlišná od skutečnosti, kterou zákazník vidí. Topologie odkazovala na prvky, které již neexistovaly, nebo pomocí špatného propojení, část sítě přemostila. V průběhu hledání příčiny problému jsem přišel na fakt, že ta topologie, která se jevila jako špatná, protože přemostila část sítě, odpovídala TOPu, tedy hlavní verzi, a že v plánované verzi sítě byla špatně vytvořena nebo dokonce nevytvořena vůbec.

Problém tohoto typu jsem musel opravit tak, že jsem na základě určitých pravidel ručně vytvořil topologické prvky, tedy linky a body, správně je mezi sebou propojil a vytvořil relaci mezi geometrickými prvky, které zákazník měl zavedené v plánované verzi elektrické sítě.

7.3 Investigace problémů s verzemi

Jak jsem již zmínil, v průběhu investigací problémů s elektrickou sítí, jsem se musel naučit pracovat s ESRI verzemi a musel jsem se snažit pochopit, jak tato hierarchie funguje. Musel jsem plně pochopit, jak propisování změn z jednotlivých podřazených verzí do verzí nadřazených funguje, abych mohl komplexně investigovat a opravovat vzniklé chyby.

Na základě určitých již výše zmíněných mechanik, které aplikace společně s ESRI versions implementuje, jsem byl schopen hledat komplexní chyby mezi verzemi, hledat v historii verzí a editovacích sezeních, tedy v módech pro editaci verzí, procházet verzovaná data a hledat relace mezi nimi, hledat komplikované chyby na základě několika specifických tabulek a určovat tak možné příčiny jednotlivých problémů.

Vzhledem k tomu, že procesy, které se dějí s verzemi, jsou velice složité, existuje zde mnoho podmínek, které společně zabraňují poškození hierarchie verzí, verzovaných dat a tak dále. Abych byl schopen přiblížit čtenáři jeden z problémů, které jsem měl na starosti investigovat a opravit, rád bych popsal jeden z mnoha důležitých procesů, který aplikace NIS zákazníkovi nabízí.

V rámci editovacího sezení, editačního módu, může pracovník různě modifikovat síť, kreslit nové prvky, tvořit nová vnitřní schémata, plánovat různé údržby a tak dále. V momentě, kdy je spokojen se svou prací nebo na konci dne, by měl alternativu propsat do plánové verze. V momentě, kdy je plán v pořádku, schválen a popřípadě realizován, provádí se POST plánu do hlavní verze. POST plánu však není povolen v momentě, kdy má podřazenou verzi. Stávalo se však, že zákazník žádnou alternativu ve výpisu neviděl, tudíž by POST měl proběhnout. Následovalo zakládání problémů, kterých jsem dostal na starost.

Musel jsem se tedy připojit na zákaznická prostředí a pokusit se chybu nalézt. Na základě nabytých zkušeností se strukturou databáze a s aplikačním prostředím, jsem začal investigovat jednotlivé relevantní tabulky. Po krátké analýze jsem zjistil, že plán, pro který nejde provést POST, nemá žádný aplikační záznam pro alternativu, ale existuje pro něj ESRI verze, kterou zákazník nemá možnost v aplikaci vidět, kvůli chybějícího záznamu pro aplikaci.

V rámci investigace a následné konzultace s týmovými seniorními specialisty se mi povedlo identifikovat chybu. V důsledku konzultace a zjištění, že takový POST verzí se provádí pomocí transakce, aby se zachovala integrita dat, mě napadlo, že může jít o havarovanou transakci, která měla za úkol smazat ESRI verzi. Zjistil jsem, že záznam o verzi pro aplikaci se maže ještě dříve, než se spustí transakce na provedení změn v rámci ESRi verzí. Za velkého vytížení nebo v případě databázových zámků ve specifických chvílích, transakce havarovala, ale záznam o verzi, který mohl vidět zákazník v aplikaci již neexistuje.

7.4 Oprava nevalidních verzí

Na základě předchozí investigace a díky nalezení příčiny jsem byl schopen navrhnout opravu. Verze, které by neměly existovat, bylo třeba smazat, aby zákazník mohl provádět POST nebo RECONCILE.

Po konzultaci s kolegy, jsem zjistil, že existuje procedura od ESRI, která dokáže smazat verzi včetně veškerých relací a kterou mohu spustit manuálně.

Po správné identifikaci chybné verze pomocí SQL relačního dotazu se mi povedlo smazat ESRI verzi a mohl jsem dát vědět zákazníkovi, že plán je připraven na POST. Nicméně v rámci tohoto problému mě napadlo zkontrolovat kompletně všechny verze, které se v databázi nacházejí, a minimalizovat tak, že zákazník bude mít problém v dalším případě ze stejného důvodu. Pomocí kurzoru, modifikace dotazu, který jsem již použil, a znalosti procedury na mazání verzí, jsem identifikoval a následně smazal všechny takové podobné verze.

Abych si pojistil, že v případě, že se takový problém objeví znovu buď z jiného důvodu a nebo do doby, než bude mít zákazník dodán hotfix, budeme první, kdo o tom ví, přidal jsem kontrolu do monitorovací ASP.NET Core aplikace, která na zákaznických prostředích běží. Po dobu několika následujících týdnů jsem každé ráno kontroloval nástěnku naší monitorovací aplikace, abych tak odhalil případné problémy.

```
SELECT * FROM [ESRI_VERSIONS]
   WHERE [VERSION_NAME] NOT IN (SELECT [VERSION_NAME] FROM [PLAN_ALTERNATIVES])
      AND parent_name NOT LIKE 'TOP'
      AND parent_owner LIKE '<DATABASE_OWNER>';
```

Výpis kódu 7.3: Anonymizovaný SQL dotaz k nalezení nevalidních verzí

Po dodání opravené verze se ale taková nevalidní verze sem tam objevila, takže jsem na základě získaných znalostí a spolupráce s kolegou našel další chybu v kódu. Chyba způsobila že v určitých případech, konkrétně v momentě poškozeného .zip souboru, který obsahoval XML plány a další data, kdy byla vyvolána výjimka, nebylo postaráno o již vytvořenou ESRI verzi. Ta byla vytvořena za účelem importování dat z XML plánu. Pomocí relativně jednoduché záplaty se mi povedlo chybu opravit a postarat se tak o takové verze, jež tímto způsobem vznikaly.

7.5 Odhalení transakční kolize v případě importu plánů

Vzhledem k tomu, že jsem již měl zkušenosti s podobnými problémy s importy a různými problémy s verzemi, dostal jsem na starosti odhalit jeden zapeklitý problém, který se nedařilo dlouhodobě vyřešit a dle slov kolegů existoval již téměř od počátku užívání aplikace zákazníky. Objevoval se však zřídka kdy a nedařilo se jej cíleně replikovat. Jednalo se chybu v importování plánů, kdy ve zdánlivě nezávisle na sobě chvílích import, který třeba trval několik desítek minut, prostě havaroval.

Snažil jsem se nalézt různé relace mezi jednotlivými situacemi třeba na základě verzí, kontroloval jsem konfigurace časových limitů jednotlivých operací, ale nenašel jsem téměř nic. Nicméně napadlo mě, že problém možná nemusí být v objemu importovaných dat či v dlouho trvajících operacích při velké zátěži.

Vytvořil jsem si drobný XML plán s několika málo změnami elektrické sítě, přihlásil se do více prohlížečů na různé účty v aplikaci a spustil jsem import kopií tohoto XML plánu co nejrychleji za sebou v různých oknech přihlášen pod různými účty.

Všiml jsem si, že vždy jeden z importů havaroval. Snížil jsem počet spuštěných importů pouze na dva a na základě opakovaného importování jsem zjistil s téměř naprostou přesností, že se jedná vždy o druhý spuštěný import. Jednou z prvních věcí, co mě napadla byla, že se pravděpodobně nejedná tedy o problém s verzemi či například s časovými limity jednotlivých procesů, ale jde o jakousi kolizi mezi importy.

Po bližší investigaci i v rámci kódu aplikace, jsem přišel na to, že určité části importu se provádějí transakčně a v momentě, kdy třeba dva uživatele pracovali se stejnou tabulkou třeba různými procesy, mohlo dojít k problému, že transakce za účelem zachování určité integrity dat, havarovala. S tou transakcí samozřejmě i import. Pod dohledem mého kolegy softwarového architekta se mi povedlo některé tyto problematické operace importu a operace, které mohou způsobit kolizi, předělat a zajistit tak, aby se tento problém již neopakoval.

Oprava této chyby pro tým zejména testerů znamenala hodně, protože díky opravě bylo testování importů a dalších záležitostí mnohem efektivnější a testing si mohl být jist, že import funguje správně.

Kapitola 8

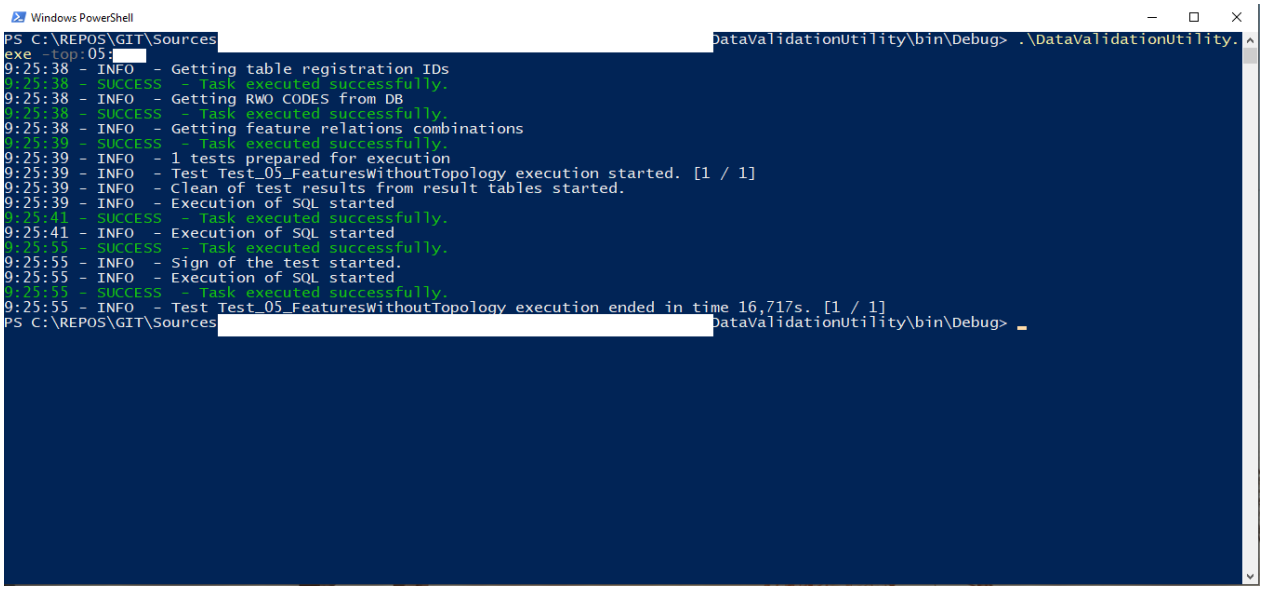
Implementace validační utility elektrické sítě

V rámci své bakalářské praxe jsem dostal na starost implementaci nástroje, která bude efektivně vyhledávat a hlásit chyby, které se v elektrické síti objeví. Hlavními požadavky na aplikaci byly maximální důraz na výkon, programátorská přívětivost a jednotlivé testy, které aplikace provádí, musejí být atomické, tedy musejí hlásit v nejlepším případě pouze jednu chybu.

Vzhledem k tomu, že předpokládaná zátěž na databázi měla být vysoká, musel jsem dbát na to, abych nedělal zbytečné operace navíc, využíval maximálně funkcionalitu SQL serveru a vše co se dalo, dělal hromadnými operacemi bez použití zbytečných kurzorů, cyklů, vnořených dotazů a tak dále.

8.1 Architektura aplikace

Pro vývoj aplikace jsem zvolil .NET Core, přičemž ovládání aplikace je realizováno skrze příkazový řádek a sérií vstupních parametrů na základě požadavků, které uživatel chce. Díky tomu může být aplikace spouštěna například pomoci Task Scheduleru v pravidelných intervalech na zákaznických prostředích.



Obrázek 8.1: Ukázka práce s Data Validation Utility

Jednotlivé testy, které aplikace implementuje, jsou psány jako vlastní třídy, kde využívám dědičnosti z abstráktní třídy **TestBase**, která obsahuje všechny metody a vlastnosti, který každá třída testu buď využívá nebo musí obsahovat. Společnými atributy jsou například kategorie testu, unikátní číslo sloužící jako identifikátor testu v rámci kategorie a detailnější popis testu sloužící jako nápověda pro uživatele nástroje.

Pro přístup do databáze jsem implementoval zvlášť třídu **TestUtilApp**, která dědí z abstraktní třídy **TestUtilBase** a se stará o exekuování SQL testů, příkazů a metod, které test obsahuje, mazání starých hlášení, načítání dat do repozitáře a tak dále.

Pro spouštění jednotlivých testů jsem využil reflexe a veškeré testy jsou načítány dynamicky ze sestavení aplikace a následná metoda **ExecuteTest()** implementující SQL test a obsahující volání důležitých metod, je volána. O tento proces se stará třída **TestRunner**, jež má na starosti, jak bylo již zmíněno, hledání odvozených typů v sestavení na základě názvu a typu třídy TestBase, a následnou exekuci testů.

V aplikaci využívám různých typů konfigurace. Pro jednotný formát logování využívám konfiguračních záznamů v databázi a pro nastavení připojení k databázi využívám klasické XML aplikační konfigurace.

8.2 Rozdělení testů elektrické sítě

Testy jsou rozděleny do třech kategorií. Geometrické testy, topologické testy a testy atributové.

Geometrické testy mají za úkol odhalit prvky s vadnou geometrií a nacházet vadné geometrické relace mezi jednotlivými prvky.

Topologické testy mají odhalit chyby v topologii elektrické sítě jako takové nebo vyhledat chyby plynoucí z relací mezi topologií a geometrickými, tedy elektrickými prvky jako takovými.

V neposlední řadě pak testy atributové, které hledají například relace mezi tabulkami pomocí cizích klíčů a jsou schopny tak odhalit špatné relace mezi prvky, popřípadě mohou nacházet chyby v prvcích, které jsou složeny z více částí. Pro příklad uvedu prvek s názvem hypernode, který slouží jako propojení vnitřního elektrického schématu například elektrické rozvodny a venkovní sítě. Hypernode se skládá ze dvou bodů a linkou mezi nimi.

8.3 Struktura databázových tabulek

Pro aplikaci jsem musel vytvořit své vlastní tabulky v geodatabázi díky nutnosti v případě potřeby provádět testy pouze nad určitou částí sítě dané pomocí rozsahu zadaným koordináty.

Vytvořil jsem tedy databázový skript, který byl zodpovědný za vytvoření třech tabulek, kde dvě z nich byly geometrické. Kromě samotného vytvoření byl skript taktéž zodpovědný za registraci tabulek do geodatabáze, aby bylo možné využívat funkcí a procedur od společnosti ESRI. Díky registraci jsem mohl využívat například jednotného přístupu v rámci databáze k ID unikátních záznamů.

Zvolil jsem dvě geometrické tabulky kvůli toho, aby bylo možné oddělovat linkové objekty a objekty bodové. Příkladem bych rád uvedl již zmíněný rozpojovač nebo jistič v roli bodového objektu a kabeláž k nim připojenou v roli linkového objektu. Kromě informací o jaký objekt se přesně jedná, každý záznam nese informaci o tom, jaký test jej přesně nahlásil a kdy jej nahlásil. Díky tomu je možné přesně identifikovat chyby na základě informací o testech a je okamžitě patrné, kdy byly chyby nalezeny.

Třetí tabulka slouží jako záznamový arch, kde se každý test spuštěný v určitou dobu podepíše včetně své unikátní identifikace na základě kategorie a čísla, časového razítka a dodatečného popisu, co vlastně test nachází.

8.4 Příklady komplexnějších testů včetně úryvků kódu

Jedním z prvních komplexnějších testů byl test hledající prvky se špatným napětím vzhledem k sousedícím prvkům. Zde jsem se poprvé setkal s relativně velkou zátěží testů, protože jsem prvně zvolil cyklický přístup zpracovávání záznamu po záznamu na základě dobře poskládaného SQL dotazu. Po bližší investigaci, konzultaci s mým mentorem, který zastává roli softwarového architekta, a po trochu delším hledání odpovědí na mé otázky na internetu, jsem se rozhodl skript přepsat a využívat tak plně výhod, které SQL server nabízí. Veškeré nalezené prvky jsem vložil do dočasné tabulky, určování ID jsem nedělal pomocí procedury vracející jedno unikátní ID pro každý prvek [13], ale naopak pomocí procedury, která mi vrátila počáteční ID a zarezervovala přesný počet ID pro ostatní prvky a vkládání nových záznamu do tabulek pro chyby jsem dělal hromadně. Tímto jsem byl schopen redukovat dobu trvání testu z 90 sekund na sekundy 2 při objemu dat na našich testovacích prostředích.

```
/*Create temp table to ease workload on SQL server, count results in temp table,
    use count of results to set how many IDs are needed, save to result table.*/
command += $@"
SELECT IDENTITY(INT,1,1) AS ID, * INTO #TEMP_RESULT_TABLE FROM (
<SQL Query to find invalid features>
) as temp
DECLARE @requestedIds INT = (SELECT COUNT(*) FROM #TEMP_RESULT_TABLE);
DECLARE @baseId INT;
DECLARE @obtainedIds INT;

exec i{CacheRepository.TableRegistrationIds["data_validation_result_line"]}
    _get_ids 2, @requestedIds, @baseId OUTPUT, @obtainedIds OUTPUT;

INSERT INTO [DATA_VALIDATION_RESULT_LINE] ([ID], [DEVICE_TYPE], [DEVICE_ID], [
    SCHEMA_OWNER_TYPE], [SCHEMA_OWNER_ID], [TEST_NUMBER], [SEVERITY], [TIMESTAMP],
     [LOCATION], [TEST_CATEGORY]) SELECT [ID] + @baseId - 1, {CacheRepository.
    DeviceType["link"]}, [LINK_ID], [LINK_SCHEMA_OWNER_TYPE], [
    LINK_SCHEMA_OWNER_ID], @testNumber, @severity, @timestamp, null, @testCategory
     FROM #TEMP_RESULT_TABLE;
";
```

Výpis kódu 8.1: Anonymizovaný úryvek optimalizované SQL funkce na detekci špatného napětí mezi relačními prvky

Dalším z komplexnějších testů bylo hledání elektrických prvků bez topologie, kde jsem chtěl využít dotazu složeného pomocí UNION operátoru. Vzhledem k tomu, že bodových prvků bylo téměř dvacet, zvolil jsem cestu dynamického skládání SQL dotazu pomocí názvů jednotlivých prvků a poprvé jsem použil repozitář sloužící k ukládání dat, které by mohl využívat každý test. Díky tomuto přístupu jsem byl schopen generovat jakýkoliv UNION SQL dotaz nezávisle na počtu prvků. Pro zahrnutí dalšího prvku do testování bylo pouze nutné přidat jeho databázový název do seznamu názvů prvků. Díky tomu jsem tak poskytl relativně efektivní možnost úpravy testu vzhledem k tomu, že si vývojář může zvolit, které prvky chce testovat.

```
command += $@"SELECT IDENTITY(INT, 1, 1) AS ID, * INTO #TEMP_RESULT_TABLE FROM(";

        foreach (var tableName in <cached_feature_table_names>)
        {
            if (tableName.Equals(<last_cached_feature_table_name>))
            {
                command +=
                    $@"SELECT f.[ID] as FEATURE_ID, <device_type> as DEVICE_TYPE
                        FROM [{tableName}] f
            LEFT JOIN [LINKS] l ON f.[ID] = l.[DEVICE_ID] AND l.[DEVICE_TYPE] =
                 <device_type>
            WHERE l.[ID] IS NULL) as temp;";
            }
            else
            {
                command += $@"SELECT f.[ID] as FEATURE_ID, <device_type> as
                    DEVICE_TYPE FROM [{tableName}] f
               LEFT JOIN [LINKS] l ON f.[ID] = l.[DEVICE_ID] AND l.[DEVICE_TYPE]
                    = <device_type>
               WHERE l.[ID] IS NULL

           UNION

           ";
            }
        }
```

Výpis kódu 8.2: Anonymizovaný úryvek kódu demonstrující dynamické generování UNION SQL dotazu 1

A neposledním testem, který stojí za zmínění je test na relace databázových záznamů na základě odkazů pomocí cizích klíčů a názvů sloupců. Díky tomu, že je geodatabáze koncipována tak, aby každá tabulka sloužící pro jeden prvek, ku příkladu měděná lyžina, která odkazuje na prvek jiný, třeba na majitele schématu, tedy rozvodna například, měla v názvu sloupce, kde jsou uloženy cizí klíče, název odkazovaného prvku, mohl jsem pomocí složitého SQL dotazu dynamicky vyhledat veškeré relace a na základě těchto nalezených relací je pak pro každý prvek testovat.

```
command += $@"SELECT IDENTITY(INT, 1, 1) AS ID, * INTO #TEMP_RESULT_TABLE FROM(";

            foreach (var relation in <cached_feature_relations>)
            {
                if (relation.Equals(<last_cached_feature_relation>))
                {
                    command += $@"SELECT f.[ID] AS FEATURE_ID, <device_type> AS
                        DEVICE_TYPE FROM [{relation.Item1}_evw] as f
                  LEFT JOIN [{relation.Item2}_evw] rel ON rel.[ID] = f.[{relation.
                      Item2}]
                    WHERE (f.[{relation.Item2}] IS NOT NULL OR f.[{relation.Item2}]
                        != 0) AND rel.[ID] IS NULL) as temp;";
                }
                else
                {
                    command += $@"SELECT f.[ID] AS FEATURE_ID, <device_type> AS
                        DEVICE_TYPE FROM [{relation.Item1}_evw] as f
                  LEFT JOIN [{relation.Item2}_evw] rel ON rel.[ID] = f.[{relation.
                      Item2}]
                    WHERE (f.[{relation.Item2}] IS NOT NULL OR f.[{relation.Item2}]
                        != 0) AND rel.[ID] IS NULL

                  UNION

                  ";
                }
            }
```

Výpis kódu 8.3: Anonymizovaný úryvek kódu demonstrující dynamické generování UNION SQL dotazu 2

Kapitola 9

Závěr

9.1 Použité znalosti získané v průběhu studia

V průběhu vykonávání mé odborné stáže jsem využil celou řadu nabytých znalostí z předmětů, kterými jsem si postupně prošel v průběhu studia. Rád bych zmínil znalosti z předmětů Databázových a informačních systémů, díky kterým jsem byl schopen efektivně tvořit komplexní SQL operace.

Dále Algoritmy a předměty s programovacími jazyky, jako jsou Programování I a II, Programovací jazyky II a v neposlední řadě Architektura technologie .NET, které mi pomohly postupem času porozumět složité architektuře aplikací, na kterých jsem pracoval.

Další výhodou pro mě bylo absolvování předmětu Vývoj informačních systémů či Úvod do softwarového inženýrství, kde jsem se setkal s návrhovými vzory a s procesy spojenými s návrhem aplikací.

9.2 Chybějící znalosti

Z počátku mé stáže mi chyběly znalosti o testování aplikací, které jsem musel obsáhnout v co nejkratším čase, abych byl schopen odvádět kvalitní práci. Díky faktu, že programovat v rámci stáže, jsem začal až později, jsem měl štěstí, že jsem své znalosti mohl doplňovat v průběhu na základě znalostí, které mi poskytlo studium. Rád bych ale však zmínil práci s ESRI verzemi, ArcGIS nástroje a geodatabázi.

9.3 Dosažené výsledky v průběhu odborné praxe

Byl jsem schopen být plnohodnotným členem vývojářského týmu a pomáhat tak s otestováním komplexních aplikací. V průběhu praxe jsem opravil mnoho chyb a implementoval několik vylepšení různých částí aplikací. V neposlední řadě jsem implementoval validační nástroj elektrické sítě a poskytl tak NMA a zákazníkům velmi silný nástroj k validaci elektrické sítě.

9.4 Zhodnocení

Jsem velice rád, že jsem měl možnost plnit stáž a odbornou praxi v TietoEVRY v NMA, protože jsem měl možnost spolupracovat se specialisty ve svém oboru a pracovat tak na velmi komplexní aplikaci. Do té doby jsem si nedovedl představit, jak takový vývoj v realitě probíhá a jak může být aplikace, jako je NIS, tak obrovská a složitá.

Odborná praxe pro mě byla velkým přínosem, protože jsem se naučil celou řadu nových věcí a osvojil si tak procesy spojené s vývojem aplikace. Byl jsem schopen zastávat roli softwarového vývojáře a investigovat tak komplexní chyby a navrhovat jejich řešení. Implementoval jsem různé kontroly a nástroje, abych zajistil vyšší efektivitu práce a pomohl s investigací jednotlivých chyb v datech.

Pomáhal jsem ostatním členům týmu s různými procesy, měl jsem dokonce možnost zaučit nové členy testerské části týmu a předat jim tak své znalosti, které jsem v průběhu nabyl.

Mou odbornou praxi jsem vykonával s radostí a s vědomím, že je dokonalou přípravou na budoucí uplatnění v oboru. Díky ní jsem byl schopen najít své slabosti a silné stránky a na základě toho se dále rozhodnout, kam budu svůj vývoj směřovat.

Literatura

1. *TietoEVRY: A leading digital services and software company.* Dostupné také z: https://www.tietoevry.com/en/about-us/our-company/.
2. *TietoEVRY: Who we serve.* Dostupné také z: https://www.tietoevry.com/en/industries/.
3. *TietoEVRY: Blog.* Dostupné také z: https://www.tietoevry.com/en/blog/.
4. *TietoEVRY: Ostravská pobočka TietoEVRY patří mezi nejlepší zaměstnavatele v České republice.* 2020. Dostupné také z: https://www.tietoevry.com/cz/novinky/vsechny-zpravy/nejnovejsi-zpravy/2020/tietoevry-nejlepsi-zamestnavatel/.
5. HUGHES, Stedman. *TechTarget | SearchSQLServer: Microsoft SQL Server.* 2019. Dostupné také z: https://searchsqlserver.techtarget.com/definition/SQL-Server.
6. HUGHES. *TechTarget | SearchSQLServer: T-SQL (Transact-SQL).* 2019. Dostupné také z: https://searchsqlserver.techtarget.com/definition/T-SQL.
7. *esri: About ArcGIS.* 2021. Dostupné také z: https://www.esri.com/en-us/arcgis/about-arcgis/overview.
8. *WiX Toolset: Wix Toolset Documentation.* Dostupné také z: https://wixtoolset.org/documentation/.
9. *Microsoft: What is .NET?* 2021. Dostupné také z: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet.
10. *Microsoft | Docs: Entity Framework documentation.* 2021. Dostupné také z: https://docs.microsoft.com/ef/.
11. WALAT. *TechTarget | SearchWindowsServer: Microsoft Windows Server OS (operating system).* 2017. Dostupné také z: https://searchwindowsserver.techtarget.com/definition/Microsoft-Windows-Server-OS-operating-system.
12. *esri: Reconcile and post edits to a traditional version.* 2021. Dostupné také z: https://pro.arcgis.com/en/pro-app/latest/help/data/geodatabases/overview/reconcile-and-post-edits-to-a-version.htm.

13. *esri: Next RowID*. 2021. Dostupné také z: https://desktop.arcgis.com/en/arcmap/latest/manage-data/using-sql-with-gdbs/next-rowid.htm.